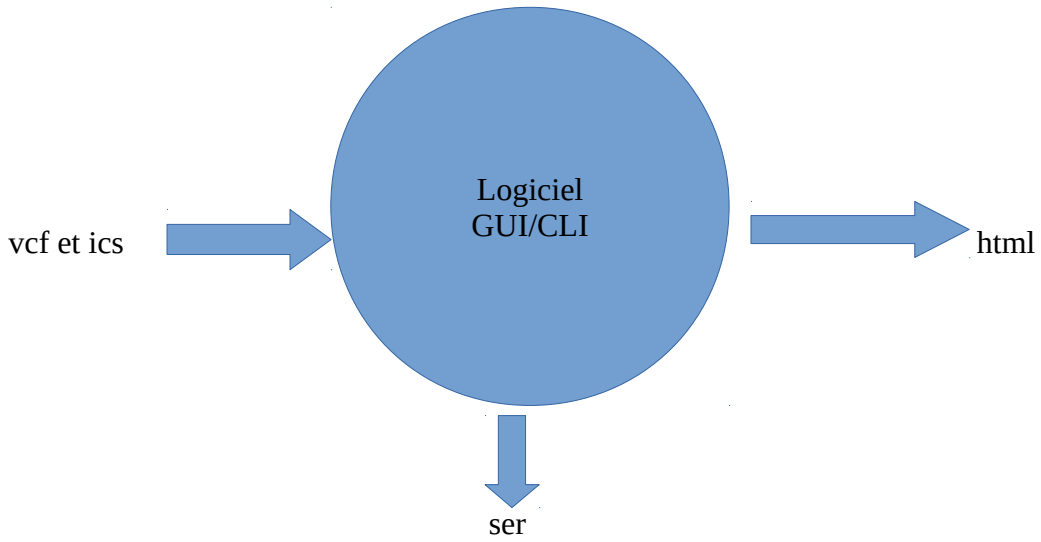
UNIVERSITÉ
de Cergy-Pontoise

Rapport de projet JAVA & POO

« De vCalendar à h-event »



Projet réalisé dans le cadre du module POO 2018/2019.

Equipe projet composée de Cazé Virginie et de Szathmary David, TD C, sous l'enseignement de Bourdon Jean-Luc et Lemaire Marc.

Table des matières

Introduction :

Le projet de cette année consiste à créer une application exécutable en console (CLI) et en graphique (GUI) permettant de manipuler certains types de fichiers : les .vcf et les .ics.

Les manipulations réalisées sont : le parcours d'un dossier quelconque et ses sous-dossiers pour lister les fichiers vcf et ics, la sauvegarde des fichiers avec la sérialisation et la génération d'un fragment HTML valide à partir de ces fichiers.
Nous avons commencés par séparer le problème en deux parties : une pour chaque type de fichier. Mais nous nous sommes finalement orientés vers la liste des tâches proposée sur l'UCP, beaucoup plus claire que notre diagramme initial.

Dans ce rapport, nous allons vous détailler notre démarche pour réaliser ce projet.

Répartition des tâches (finale) & planning :

Virginie	-classes manipulant les fichiers .vcf -méthode de recherche dans une arborescence de dossiers -classes utilitaires annexes (Filtre, exception, interface, etc.) -création de l'interface graphique et de ses fonctionnalités
David	-création du main du mode console avec gestion des paramètres d'exécution -classes manipulant les fichiers .ics

Voici un petit récapitulatif de notre planning :

-la 1ere semaine, nous avons décortiqué le sujet pour être sûrs de pas être hors-sujet. Puis nous avons commencé par les fichiers .vcf. Virginie a codé la première classe de stockage pour pouvoir les informations de nos fichiers dans des attributs. Puis nous avons codé la classe permettant de pouvoir afficher la liste des fichiers vcf et ics dans une arborescence de dossiers.

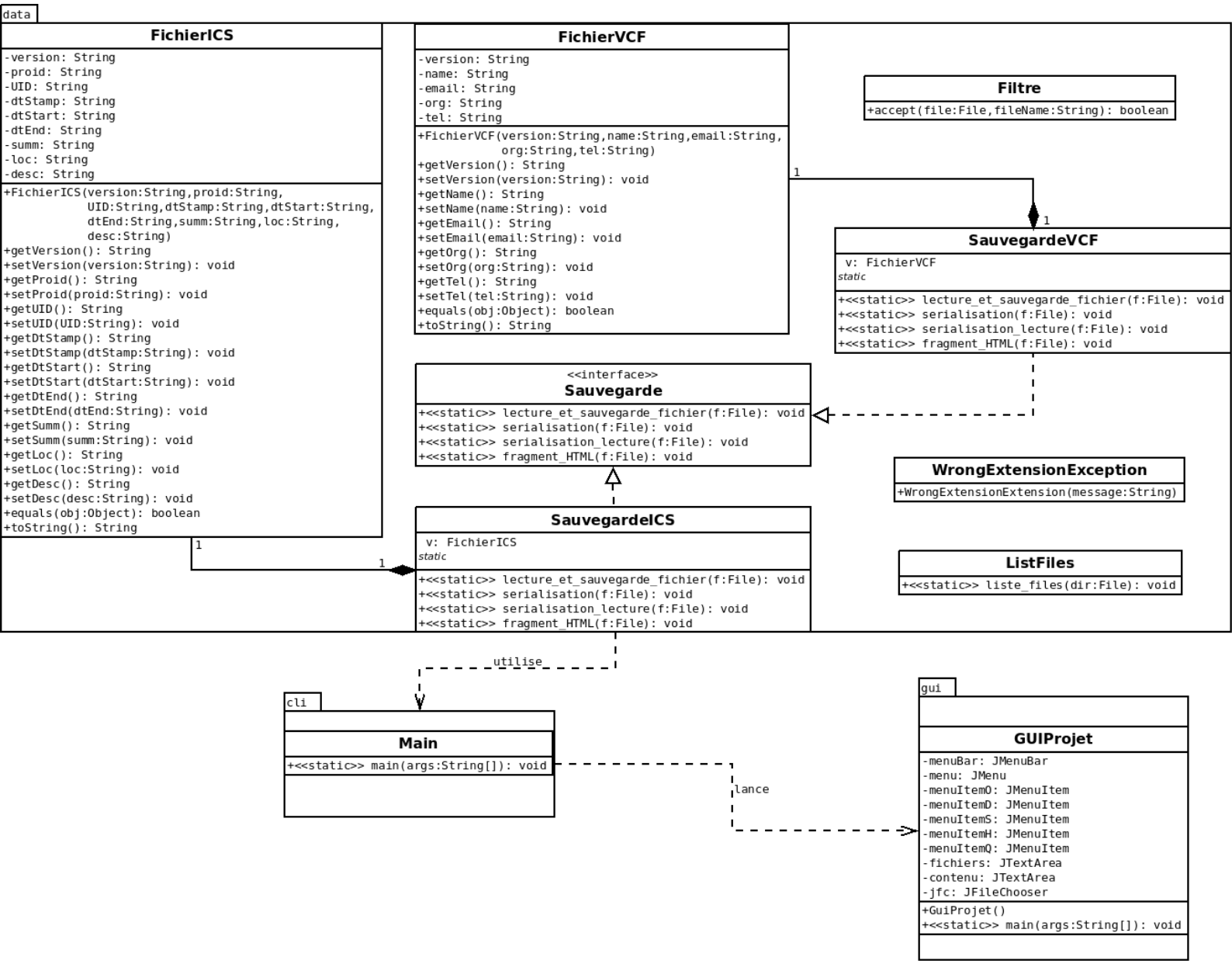
-les semaines suivantes, nous avons avancé vite, ne trouvant pas de difficulté particulière. Nous avons codé toutes les manipulations sur fichiers .vcf (hormis la génération HTML) et David a travaillé sur les classes de stockage des fichiers ics et de leurs manipulations.

-la semaine suivante, David, qui avais commencé le mode console, put quasiment le finir. Virginie s'attela à créer une méthode pour générer un fichier HTML à partir d'un vcf, ce qui posa les premières difficultés car il nous fallait alors un objet de type FichierVCF pour permettre d'utiliser les infos de ces fichiers. Nous avons résolu cela en combinant la méthode de sauvegarde sur un objet et la génération HTML. Le fragment HTML ne permettait alors pas de créer une page html fonctionnelle.

-David fit de même pour les ics, et termina le mode console alors que Virginie construit le prototype du mode graphique ainsi que ses premières fonctionnalités (ouverture d'un fichier, listing des fichiers d'un dossier et sauvegarde).

-le point d'avancement passé, l'équipe projet put terminer le mode graphique.

UML :


data
FichierICS
-version: String
-proid: String
-UID: String
-dtStamp: String
-dtStart: String
-dtEnd: String
-summ: String
-loc: String
-desc: String
+FichierICS(version:String,proid:String, UID:String,dtStamp:String,dtStart:String, dtEnd:String,summ:String,loc:String, desc:String)
+getVersion(): String
+setVersion(version:String): void
+getProid(): String
+setProid(proid:String): void
+getUID(): String
+setUID(UID:String): void
+getDtStamp(): String
+setDtStamp(dtStamp:String): void
+getDtStart(): String
+setDtStart(dtStart:String): void
+getDtEnd(): String
+setDtEnd(dtEnd:String): void
+getSumm(): String
+setSumm(summ:String): void
+getLoc(): String
+setLoc(loc:String): void
+getDesc(): String
+setDesc(desc:String): void
+equals(obj:Object): boolean
+toString(): String
FichierVCF
-version: String
-name: String
-email: String
-org: String
-tel: String
+FichierVCF(version:String,name:String,email:String, org:String,tel:String)
+getVersion(): String
+setVersion(version:String): void
+getName(): String
+setName(name:String): void
+getEmail(): String
+setEmail(email:String): void
+getOrg(): String
+setOrg(org:String): void
+getTel(): String
+setTel(tel:String): void
+equals(obj:Object): boolean
+toString(): String
Filtre
+accept(file:File,fileName:String): boolean
SauvegardeVCF
v: FichierVCF
static
+<<static>> lecture_et_sauvegarde_fichier(f:File): void
+<<static>> serialisation(f:File): void
+<<static>> serialisation_lecture(f:File): void
+<<static>> fragment_HTML(f:File): void
<<interface>>
Sauvegarde
+<<static>> lecture_et_sauvegarde_fichier(f:File): void
+<<static>> serialisation(f:File): void
+<<static>> serialisation_lecture(f:File): void
+<<static>> fragment_HTML(f:File): void
WrongExtensionException
+WrongExtensionExtension(message:String)
SauvegardeICS
v: FichierICS
static
+<<static>> lecture_et_sauvegarde_fichier(f:File): void
+<<static>> serialisation(f:File): void
+<<static>> serialisation_lecture(f:File): void
+<<static>> fragment_HTML(f:File): void
ListFiles
+<<static>> liste_files(dir:File): void
1
1
1
1
utilise
cli
Main
+<<static>> main(args:String[]): void
lance
gui
GUIProjet
-menuBar: JMenuBar
-menu: JMenu
-menuItemO: JMenuItem
-menuItemD: JMenuItem
-menuItemS: JMenuItem
-menuItemH: JMenuItem
-menuItemQ: JMenuItem
-fichiers: JTextArea
-contenu: JTextArea
-jfc: JFileChooser
+GuiProjet()
+<<static>> main(args:String[]): void

Spécificités de réalisation :

1)Le Filtre :

Pour parcourir l'arborescence de dossiers et récupérer certains types de fichiers, nous avons préféré utiliser une classe Filtre implémentant l'interface FilenameFilter, prévu à cet effet. Ceci aillant pour but d'organiser plus proprement le code.

```
/**
 * Filtre implemente l'interface FilenameFilter qui permet de filtrer les
fichiers par leurs noms. Ici on utilise la fin du
 * nom de fichier, aka l'extension.
*/
public class Filtre implements FilenameFilter{

	/* (non-Javadoc)
	 * @see java.io.FilenameFilter#accept(java.io.File, java.lang.String)
	 */
	public boolean accept(File file, String fileName) {
	   return fileName.endsWith(".vcf") || fileName.endsWith(".ics");
	}
}
```

Code source de la classe Filtre

2)L'interface Sauvegarde :

Dans le même but, nous avons crée une interface « Sauvegarde » qui rassemble les méthodes communes de manipulations sur fichiers(lecture & affichage, sérialisation et lecture de la sérialisation + génération HTMl), qui sont redéfinies dans les classes « SauvegardeVCF » et « SauvegardeICS ».

3)Les classes de stockage :

Pour pouvoir générer le HTML et le fichier de sauvegarde binaire, il a fallu que nous définissons des classes de stockage basiques. Lors de la lecture d'un fichier pour l'affichage, nous récupérons chaque donnée et la stockons dans un objet d'un des deux types de fichiers. Cette sauvegarde est primordiale pour la sérialisation.

```
if(line.startsWith("N:")) {
                                        String name = line.substring(2);
                                        v.setName(name);
                                }
```

Code source d'un fragment de la classe SauvegardeVCF, illustrant la sauvegarde sur objet

```
/** Methode permettant de serialiser l'objet de type FichierVCF dans un fichier
binaire. Le nom du fichier a produire est
	 * passe en parametre par l'utilisateur.
	 * @param f
	 * @see #serialisation_lecture(File)
	 */
	public static void serialisation(File f) {
```

```
            try {
                  if(f.getName().endsWith(".ser")) {
                  f.createNewFile();
                  FileOutputStream fos = new FileOutputStream(f);
                  ObjectOutputStream oos = new ObjectOutputStream(fos);
                  oos.writeObject(v);
                  fos.close();
                  oos.close();
                }
            } catch (IOException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
      }
```

Code source de la méthode de sérialisation sur fichier binaire d'un fichier VCF à partir d'un objet

4)Création du mode console :

Pour créer le CLI, David a d'abord crée un squelette de ce dernier avec différents arguments, qui sont demandés dans le sujet. Puis nous avons appeler chaque méthode nécessaire selon la fonctionnalité demandée.

Argument & Fonctionnalité :	**Méthodes appelées :**
aucun	Aucune (affiche les arguments possibles et leur utilité)
-d (liste fichiers)	ListFiles.liste_files(File f)
-i (affiche le contenu du fichier)	SauvegardeVCF.lecture_et_sauvegarde_fichier(f) SauvegardeICS.lecture_et_sauvegarde_fichier(f)
-o (sauvegarde le fichier avec la sérialisation)	SauvegardeVCF.serialisation(f) SauvegardeICS.serialisation(f)
-h (génère le fichier .html)	SauvegardeVCF.fragment_HTML(f) SauvegardeICS.fragment_HTML(f)
-gui (lancer la GUI)	new GUIProjet()

Tableau des fonctionnalités du mode console

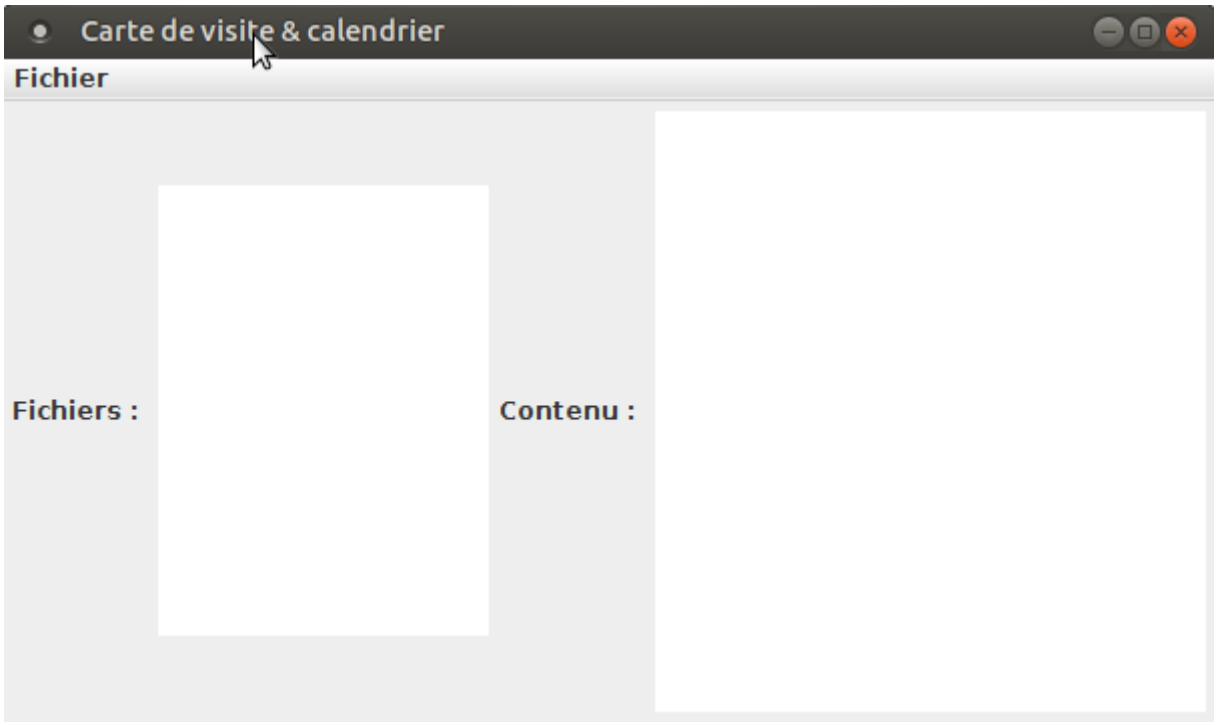
Pour utiliser les arguments et les gérés, nous parcourus le tableau args de la méthode main. Toute saisie d'argument non reconnu n'est pas gérée. Pour récupérer la saisie utilisateur, nous utilisons des attributs String qui vont récupérer la valeur d'une des cases du tableau args selon le nombre d'arguments. A noter que « -i » doit être utilisé dès que l'on veut entré un fichier en paramètre.

5)Création du mode graphique :

Le mode graphique a d'abord été crée sous la forme d'une maquette puis codé sans les fonctionnalités une fois l'équipe projet d'accord. Comme toute GUI classique, elle fut crée à l'aide d'une JFrame et de JComponents sur lesquels furent ajoutés des ActionListener.

Manuel utilisateur de la GUI :

Interface graphique :



Dans la GUI, l'utilisateur a accès à un menu déroulant « Fichier » lui offrant diverses fonctionnalités :

-Ouvrir : permet à l'utilisateur de choisir le fichier à afficher dans la zone contenu. Le contenu du fichier ouvert sera uniquement affiché si le fichier est de type .vcf/ics/ser.

-Récupérer chemins : permet à l'utilisateur d'afficher la liste des fichiers vcf et ics se trouvant dans un dossier qu'il peut sélectionner. Le chemin absolu du dossier choisi et les noms des fichiers s'y trouvant sera affiché si ces fichiers existent. Tout ceci est affiché dans la zone « fichiers » qui est non éditable contrairement à « contenu ». Note : cet option ne permet pas de choisir de fichier, uniquement des dossiers.

-Sauvegarder : permet à l'utilisateur d'enregistrer le contenu de « contenu » dans un fichier vcf/ics/ser. Cela lui permet donc d'enregistrer ses modifications.

-Générer HTML : transforme la zone contenu en code HTML et permet à l'utilisateur de l'enregistrer sous forme d'un fichier de type HTML

-Quitter : permet de choisir si on veut quitter le logiciel ou non

Remarque : si l'utilisateur entre un type de fichier non attendu (ex : txt) le logiciel n'enregistrera rien.

JComponents utilisés : Jmenu, JmenuItem, Jlabel, JtextArea,

Améliorations possibles (liste non exhaustive) :

Bien que notre logiciel respecte le cahier des charges et le sujet, il est nécessaire de noter que quelques améliorations peuvent être effectuées :

-possibilité de créer une page HTML complète fonctionnelle et non un simple fragment en rajoutant les balises head et body

-amélioration graphique : utilisation d’un autre Layout, addition d’une barre d’outils JtoolBar, fenêtre d’aide avec potentiellement le manuel utilisateur...

-simplification de la génération HTML

-conversion automatique d’un fichier texte en vcf/ics pour minimiser les erreurs utilisateur et rendre le logiciel plus maniable

Conclusion :

Points forts :

-interface graphique intuitive

-sauvegarde des vcf, ics et ser

-possibilité de connaître les fichiers d’un dossiers

-transforme du texte en HTML

-possibilité d’ouvrir la GUI depuis le CLI

-traitement total du sujet

Points faibles :

-fichiers vcf et ics non testés sur contact/calendrier

-tout les points cités dans améliorations : GUI peu poussé niveau design, page HTML non complète, etc.

-l’utilisateur a accès à seulement 3 types de fichiers utilisables